```
1: #ifndef PILAFILASIMPLE_H_INCLUDED
2: #define PILAFILASIMPLE_H_INCLUDED
3: #endif // PILAFILA_H_INCLUDED
4:
5: typedef struct
6: {
7:     int dato;
8:     struct Pila *siguiente;
9: }Pila;
10:
11: Pila *inicPila();
12: Pila *apilar(Pila *pila, int dato);
13: int tope(Pila *pila);
14: int pilaVacia(Pila *pila);
15: Pila * leer(Pila *pila);
16: int desapilar(Pila **pila);
17: void mostrarPila(Pila *pila);
18:
19:
20:
21:
22:
23:
24:
25:
26:
27:
28:
29:
30:
31:
32:
33:
34:
35:
36:
37:
38:
39:
40:
41:
42:
43:
44:
45:
46:
47:
48:
49:
50:
51:
52:
53:
54:
55:
56:
57:
58:
59:
60:
61:
62:
63:
64:
65:
66:
67:
68:
69:
```

```
 70: #include "PilaListaSimple.h"
 71: #include <stdio.h>
 72:
 73: Pila *inicPila()
 74: {
 75:     return NULL;
 76: }
 77:
 78: //DEVUELVE 1 O 0 SEGUN ESTE VACIA O NO
 79: int pilaVacia(Pila *pila)
 80: {
 81:     int vacia=0;
 82:     if(pila==NULL)
 83:     {
 84:         vacia=1;
 85:     }
 86:     return vacia;
 87: }
 88:
 89: Pila *apilar(Pila *pila, int dato)
 90: {
 91:     Pila *nuevo=(Pila*)malloc(sizeof(Pila));
 92:     nuevo->dato=dato;
 93:     nuevo->siguiente=NULL;
 94:
 95:     if(pila==NULL)
 96:     {
 97:         pila=nuevo;
 98:     }
 99:     else
100:     {
101:         Pila *aux=pila;
102:         while(aux->siguiente!=NULL)
103:         {
104:             aux=aux->siguiente;
105:         }
106:         aux->siguiente=nuevo;
107:     }
108:     return pila;
109: }
110:
111: int tope(Pila *pila)
112: {
113:     return pila->dato;
114: }
115:
116: //PIDE UN DATO POR CONSOLA Y APILA
117: Pila * leer(Pila *pila)
118: {
119:     int dato;
120:     printf("\nIngrese un numero entero para cargar a la pila: ");
121:     scanf("%d",&dato);
122:     pila=apilar(pila,dato);
123:     return pila;
124: }
125:
126:
127:
128:
129:
130:
131:
132:
133:
134:
135:
136:
137:
138:
```

```
139: int desapilar(Pila **pila)
140: {
141:     //LO IDEAL SERIA ES QUE SE MANEJE CON PUNTEROS, ES DECIR, QUE LA FUNCION
142:     //DEVUELVA UN PUNTERO A UNA PILA "DATO", POR QUE DE ESTA MANERA SI NO HAY
143:     //NINGUN NODO EN LA PILA SE PUEDE DEVOLVER NULL, Y CON NULL SE PUEDE EVALUAR
144:     // QUE NO HAY NADA.
145:     //AL FIGURAR INT EN LA FIRMA/CABECERA ESTAS OBLIGADO A DEVOLVER UN NUMERO
146:     //AL INVOCAR ESTA FUNCION HAY QUE ANTEPONER ANPERSAND (&) DELANTE DEL
147:     //PUNTERO A LA LISTA
148:
149:     //      Pila *pila=NULL;
150:     //
151:     //EJ:   int dato=desapilar(&pila);
152:     //
153:
154:     int dato=0;
155:     if((*pila)!=NULL)
156:     {
157:
158:         if((*pila)->siguiente==NULL) //SI ES EL UNICO NODO
159:         {
160:             dato=(*pila)->dato;
161:             free(*pila);
162:             (*pila)=NULL;
163:
164:         }
165:         else
166:         {
167:             Pila *aux=(*pila);
168:             Pila *ante=aux;
169:
170:             while(aux->siguiente!=NULL)
171:             {
172:                 ante=aux;
173:                 aux=aux->siguiente;
174:             }
175:             dato=aux->dato;
176:             free(aux);
177:             ante->siguiente=NULL;
178:         }
179:     }
180:     return dato;
181: }
182:
183: void mostrarPila(Pila *pila)
184: {
185:     if(pila!=NULL)
186:     {
187:         Pila *aux=pila;
188:         Pila *pilaAux=NULL;
189:
190:         //PRIMERO SE HACE UNA COPIA DEL ORIGINAL, DADO QUE AL
191:         //INVERTIR USANDO DESAPILAR SE VA A DESTRUIR
192:
193:         while(aux!=NULL)
194:         {
195:             pilaAux=apilar(pilaAux,aux->dato);
196:             aux=aux->siguiente;
197:         }
198:
199:         Pila *pilaAMostrar=NULL;
200:
201:         //INVERTIMOS LA LISTA, DESTRUYENDO pilaAux
202:
203:         while(pilaAux!=NULL)
204:         {
205:             pilaAMostrar=apilar(pilaAMostrar,desapilar(&pilaAux));
206:         }
207:
```

```
208:         //MOSTRAMOS LA LISTA INVERTIDA
209:
210:         printf("\n=============\n");
211:         printf("\n==TOPE=======\n");
212:         printf("\n=============\n");
213:         while(pilaAMostrar!=NULL)
214:         {
215:             printf("%i\n",pilaAMostrar->dato);
216:             pilaAMostrar=pilaAMostrar->siguiente;
217:         }
218:         printf("\n=============\n");
219:         printf("\n==BASE=======\n");
220:         printf("\n=============\n");
221:     }
222:     else
223:     {
224:         printf("\nLa pila esta vacia!\n");
225:     }
226: }
227:
228:
```